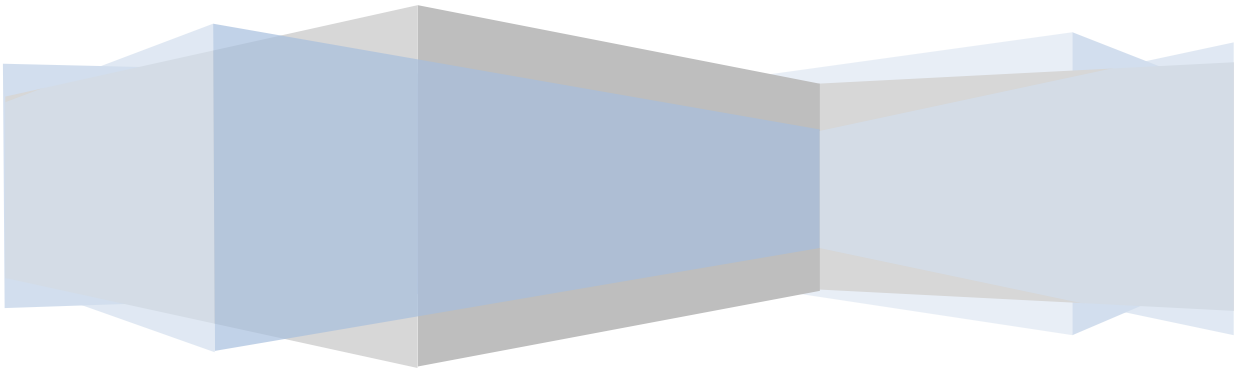
CO600 Final Year Project

Improving Content delivery in a Video-on-Demand over IP Environment



Improving Content Delivery in a Video-on-Demand over IP Environment

Name,
Affiliation,
Email Address

Abstract

During my industrial placement year, as part of my computer science degree course, I worked at Microsoft TV. Microsoft TV is a business group within Microsoft, who produce secure end to end content delivery systems for the distribution of live TV and Video on Demand aimed at the telecommunications market. The Video on demand system uses a very simple, single connection, secure HTTP content delivery method. This seemed grossly inefficient to me and I discussed it with my manager. He proposed that for my final year project I should provide a proof of concept system for a more efficient delivery mechanism.

1. Introduction

The Microsoft TV platform delivers both live TV and Video on Demand (VoD) to a set top box in the home over an IP based network, connected to a television. Live TV is broadcast using multicast based technologies. However, due to the nature of Video-on-Demand being dependant on the viewer as to when content is required, multicasting is not suitable as a delivery mechanism. Instead unicast delivery is used, which by nature is very inefficient as a single connection is required for each client. Microsoft has asked me to produce a sample implementation of an improvement on this method.

The set top box is based on a system on chip design running Windows CE (Compact Edition) with a .NET framework. The set top box also contains a 250 GB hard disk, which is used for personal video recording functions and for the caching of single play videos that are too big to play instantly. These videos are distributed by a mechanism called Download and Play, whereby the client subscribes to a category or package, and the videos are streamed to the box in the background, with the user being notified upon completion.

2. Background

Digital Broadcasting is a new and continually evolving technology. We are beginning to see the emergence of a consumer market for the next generation of Television delivered over home broadband links, IPTV (Internet Protocol Television). According to the Multimedia Research Group the number of global subscribers will grow from 4.3 million in 2005 to 72.6 million in 2011[1]. A key part of this is the ability to watch video on-demand, in the comfort of your own time.

There have been many recent developments around video delivery and as a result a number of products have come on to market in the past few years. However only one, Joost [2], comes close to the removing the need for a large server based infrastructure. Joost is still in its infancy and as a result only a beta version has been released, unfortunately it often fails to deliver video in time, or at all. The video format used in this product is also not of broadcast quality, such as required by the Microsoft TV system.

3. Aims

In this project, I aim to satisfy the following criteria:

- Improve the efficiency of delivery of video by decreasing utilization of the VoD server and increase peer to peer usage
- Retain resilience; if the peers become unavailable to deliver VoD in time, ensure this is detected and a fall back is implemented.
- Ensure playback occurs instantly, as in the existing system.

The project outcome will be a proof of concept demonstration of the above aims and will not directly integrate with the existing Microsoft TV system but may be analysed by Microsoft.

3.1 Assumptions

The Microsoft TV system has strict network requirements that need to be satisfied by the customer's network; as a result these requirements will also need to be applied to my project. Thus I will assume the following of the target platform:

- Clients are always on (This is implemented on the set top box as an active standby.)
- Home network connection is a minimum of 4Mbit downstream and 1.5Mbit upstream. (Zero Contention and minimal packet loss)
- The connection to the VoD Server can sustain the maximum available free bandwidth on the connection and has priority over other connections.

Although these assumptions are not entirely realistic as no network will be without its bottlenecks. It does reflect the Microsoft TV network requirements with regards to a Quality of Service mechanism being enforced on the home connection when the set top box is active, which improves overall connectivity. This ensures the network requirements are met at the time of use, but not when the set top box is in standby.

4. Project Management

4.1 Risk Analysis

A risk analysis was conducted that allowed identification of events that could impact on the project, both project based and external. From this analysis a table was put together identifying risk exposure for particular risks and actions to be taken to minimize impact on the project should they occur.

4.2 Quality Assurance

Quality Assurance was broken down into two factors, code quality and documentation quality.

Quality of produced code is always important, and in order to ensure that I kept the standard high I used a code analysis tool called FxCop [3]. FxCop is a community developed freely available tool that uses a rule engine based approach to scan through target code and ensure compliance with the .NET secure coding guidelines [4]. If it detects any breaches of rules it can suggest steps to fix the code.

Documentation quality was assured by using a checklist that was applied to each document and signed off on a final release. This checklist covered a variety of criteria including spelling, correct title, correct template and correct date.

4.3 Version Control

Subversion [5] was used throughout the project to keep copies of both the code and the documentation. Subversion allowed easy backing up of the code, and rolling back when required.

5. Design

5.1 Implementation Approach

Despite having only been taught Java on my degree course, I spent a substantial amount of time working with the Microsoft C# language in my placement year; this language is very similar to Java and required minimal adjustment. As this project is sponsored by Microsoft, I have chosen to use C# over Java. C# uses the .NET 2.0 framework set of libraries and includes easy integration of web services which will be of benefit to this project. It will be developed in Microsoft Visual Studio 2005 SP1.

5.2 Conceptual Design

The overall design of the system will be not that dissimilar to the existing one. It will still be based on a client to server approach; however it will use clients to supplement each other in providing the video. This will reduce loading on the server and improve efficiency.

In order to accomplish this, clients must be made aware of other clients; a web service has been implemented to satisfy this requirement. Clients must also cache video that they receive so that they can provide it to other clients.

Currently videos are single large files that are played as received by the set top box. In order to be able to support multiple simultaneous video delivery methods, the videos must be broken up in to smaller bits.

To ensure that videos start immediately, the connection to the VoD server will be used initially and then peers will contribute to and, if sufficient, take over the delivery, once they are in a suitable state to do so. Should the client sources become incapable of providing enough video to meet demand, then the connection to the VoD server will be used to ensure no interruption to delivery occurs.

5.3 Video Asset Specification

Currently the Microsoft TV platform uses an average of 10 megabytes of video for 1 minute of play; this means a 30 minute program uses 300mb.

I have decided to split videos into 15mb parts which provide 1 minute and 30seconds of video,

with 20 parts needed for a 30 minute program. Based upon this sizing a 4Mbit connection can provide up to one part every 30 seconds.

5.4 Tracking Clients

In order to provide clients with the ability to track other clients I have decided that a web service backed by a database is a reasonable approach to take.

5.4.1 Client Web service Design

The web service is coded in C# and deployed on an IIS web server that the clients can connect to. For the purposes of this project the web service will also enable clients to know about which videos exist in the system.

Web service method calls:

Client Management

AcceptHeartBeat – called by the client on a timer to signal it being active, and notify of its current IP address.

DiscoverIP – called by the client to determine the IP that is in use when facing other clients.

GetPeers – returns a list of all active clients in the system and their respective IP addresses.

Video Management

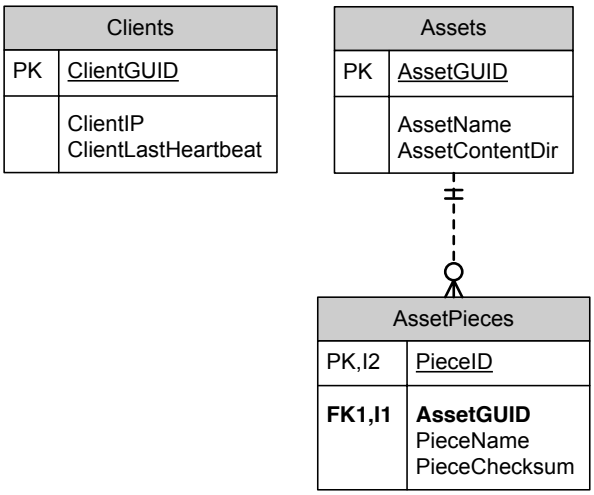
DiscoverAssetLocation – returns the address of the VoD server

GetAssets – returns a list of Assets.

GetAsset – returns detailed information about a particular asset along with a list of asset parts.

5.4.2 Client Web service Database Design

The database is a Microsoft Access database consisting of 3 tables providing the web service with relevant information.



The Clients table contains a unique identifier for the client, the clients IP address and the last date and time that the client performed a heartbeat.

The Assets table contains a unique identifier for each asset and the assets name. An asset is a complete Video.

The AssetPieces table contains a list of all asset (video) parts and their names, along with the unique identifier of the asset they belong to.

A join query is used when the web service calls GetAsset, between the Asset and the AssetParts

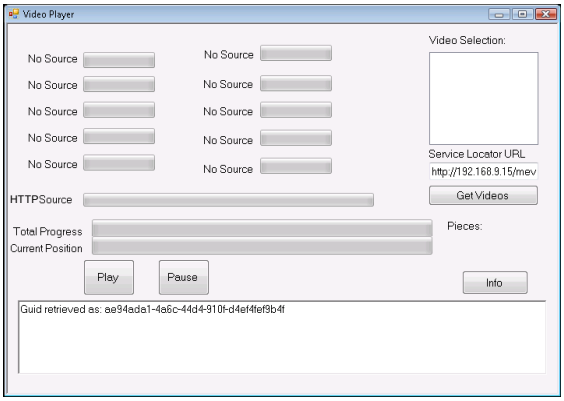
5.5 Client Design

5.5.1 UI Design

The client was designed for the Windows platform, and will run on any version of Windows that the .Net 2.0 framework is available for.

At first I was going to decode and link the video parts during playback in the client, however this proved difficult and outside of the scope of this project, as it requires writing a low level DirectX filter (which would involve a significant amount of time and research). This was due to DirectX locking the video file during playback, thus not allowing additional video parts to be linked to it.

Instead I decided to show the progress of the asset download versus the actual playing time of the asset so far, using progress bars. The revised UI is shown overleaf.



The UI has progress bars for each of the peer sources, as well as a progress bar for the existing VoD Server connection (labelled HTTP Source).

The UI also possesses a debug box at the bottom, which has proved invaluable in development of the application, and provides additional information to the user as to what actions the client is performing

One of the challenges of developing this debug box was cross thread access, which is not easily implemented in windows form controls. I used by setting a call back if the debug box was currently in use by a different thread.

5.5.2 Client Logic Design

Client connects to a user provided web service URL, upon successful connection, the client performs IP discovery using the web service. Using its retrieved IP, it then starts its video part file server bound to this IP.

A list of active clients in the system is then retrieved from the web service, fifty are selected at random and Discoverer classes are created for each of them. Each of these classes is then used to discover the held video pieces in the target client. A list of videos (or assets) is then retrieved from the web service and the list box is populated with these titles. A timer is then started that performs a heartbeat with the web service every five minutes. The client sources updater thread timer is also started.

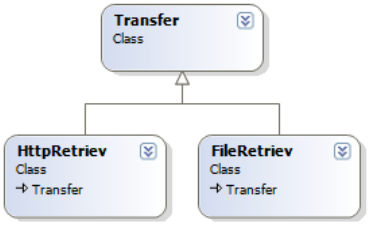
When a user then selects a video the client first checks to see if any pieces are held in the local content directory (in case of a previous partial playback) and then places the required pieces on to a stack. When the user then clicks play, the progress bar to denote position in asset begins to increment and the transfer check thread is started. This continues until the video is completely downloaded.

5.5.3 Client Network Design

The client employs two different networking designs, the first type is for communicating information and the second type is for file transfer of video parts.

The communications can then be split again in to two subsections; web service communication and inter-client communication. The web service communication involves a transactional based approach, using the web service to retrieve and post information (as previously specified) as required. This uses standard libraries to instantiate the web service as an object and then call methods on it, as you would for normal class based objects. The second type is for communicating between clients, which use their own protocol for exchanging information.

There are two different types of file transfer available, VoD Server and Peer to Peer shown below as HttpRetriev and FileRetriev respectively.



The two network classes are subclasses of Transfer, which provides methods to retrieve transfer progress and rate.

5.5.3.1 Peer to Peer Communications Protocol

Clients connect to other clients on port 51000 they then issue one or more of the following commands:

L	Request a list of all video parts held by the client
T	Send test data file
G<partname>\n	Request transfer of a video part
Q	Quit, ends the connection

5.5.3.2 Discoverer Class

The discoverer class is a key part of the networking infrastructure design. At construction it is given a target client, it can then perform two methods on it; the first gets a list of all the pieces the client holds and the second performs a speed test on the client.

The client will only call the speed test if the discoverer holds a part needed for the current video that is selected. The speed test is performed by a 1 megabyte test file being requested, and measuring the time taken for the transfer to complete. Should the client not complete the speed test within a certain time period, the test is aborted and the connection removed, this avoids unnecessary connections remaining.

The class implements the IComparable() .NET framework collection class method allowing it to be sorted when placed in a Collection. This method enables Discoverers with higher data rates to be

sorted to the top of the collection, which is used for selecting the top ten fastest.

5.5.3.3 VoD Server Transfer Design

The client has been designed to mimic the existing Microsoft TV with regards to the VoD Server connection. This involves a standard web server delivering the content over an HTTP connection. In the Microsoft TV system this is of course encrypted, however for simplicity there will be no security implemented in the client.

5.5.3.4 Peer to Peer Transfer Design

Peer to peer transfers occur over a standard TCP connection using standard stream readers and writers once a get command has been issued by the initiating client.

5.5.4 Client Sources Updater Thread

This is a thread that is called every 10 minutes to request a new list of sources from the web service these are then passed to another thread that creates a Discover class for each new client, which locates held pieces. Any clients that contain required pieces are then speed tested and placed in the ordered collection of available peers.

5.5.5 Transfer Check Thread

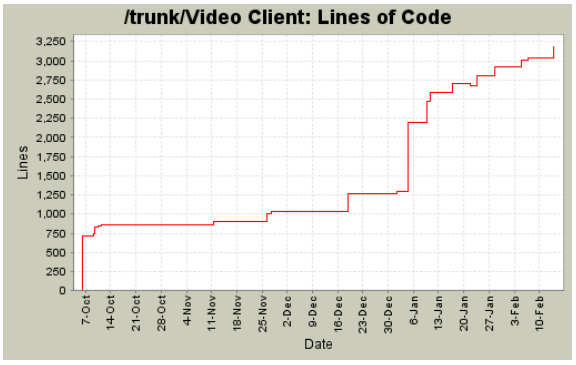
This thread checks the overall transfer progress, to see if any transfers are finished, and if so starts additional ones. Or if transfers are failed, will reallocate pieces back on the stack. It also acts as a failsafe to work out if the next piece is available for playback. If the next piece isn't available, it first checks to see if there is enough time to finish retrieving it from the peer source, if there isn't enough time it will stop the peer transfer and fall back to the VoD Server connection.

5.6 Client Cache

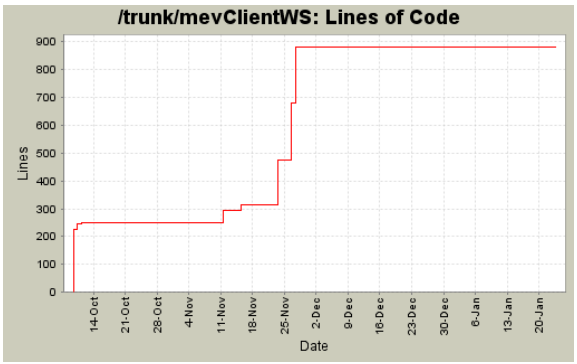
Once a part has started downloading it is placed in an incoming directory, upon successfully finishing downloading it is then moved to a content directory. This directory serves as a cache, which can be used to play back a video again without re-downloading it, or can be used to serve parts to other clients in the system.

6. Development

Development began half way through the autumn term and ended during the first half of the spring term. The following graphs show overall code development against time. This is generated based on subversion check in data, using StatSVN [6].



Client lines of code vs. Time



Web service lines of code vs. Time

As you can see on the above graphs, the development of the web service was completed before the majority of the development of the client occurred. This was necessary due to the need of a working web service for the development of the client. This was attributed primarily to the large reliance of the client upon the web service for information regarding other clients, videos and video parts.

7. Testing

Testing was split into 3 parts; Client testing, web service testing and overall system testing.

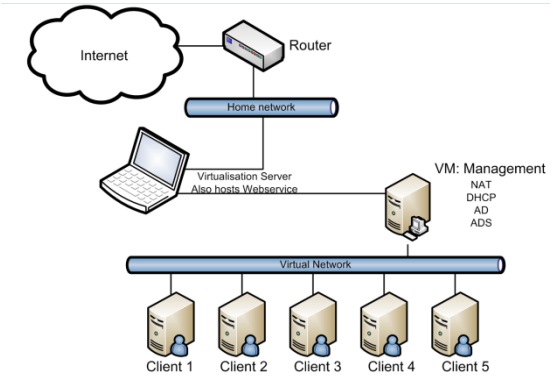
Client testing involved functional testing, of the user interface, information exchange between the web service and itself, and also inter-client communications. User interface testing covered data validation and general operation of the form controls. Web service testing ensured that data was being correctly sent, retrieved and interpreted by the client from the web service. Inter client communications were tested by opening a connection to the client using a telnet utility.

Web service testing was split into three parts, functional testing, load testing and recovery testing. Functional testing was performed to see if data was being stored and retrieved successfully from the database. Load testing was used to see at what point

the web service failed, during heavy loading. Recovery testing was performed to see if the web service automatically recovered from any errors encountered.

Testing the overall system in this project requires a sophisticated approach as multiple clients are required, each needing their own machine with differing network characteristics.

Using Microsoft Virtual Server 2005 [7], a set of virtual machines running on one machine was constructed. On each virtual machine runs a Microsoft network emulator that enables in depth tuning of the network layer. A separate virtual machine is used to manage all individual client machines, pushing out builds and modifying their network profiles. Shown below is the network architecture used during testing.



The client was tested in a variety of scenarios including one's that involved:

- No Peer Sources
- No Peer Sources with available parts
- Sudden Forced Disconnection of Peer Sources
- Slow peer sources
- Different numbers of peer sources
- Large numbers of peer sources with different connection speeds

The client successfully passed all scenarios' and successfully fell back to the VoD Server connection to satisfy in-time demand.

Load testing the web service under stress conditions however was not so successful. Web service methods that involved calls to the database were affected, when web service transactions per second approached 20, this was traced back to the OLEDB adapter not correctly behaving with my code. A solution to this would be to use a SQL server backend, as opposed to an Access flat file approach. Other than this the web service passed all the functional testing and recovery from any errors was experienced during the next web service call.

8. Conclusion

I believe this proof of concept has demonstrated an effective way of improving delivery. However I have not been able to volume test it how I would like to, and don't believe that it will scale effectively. This is due to not having a means to simulate a large amount of clients and thus gauge the level at which network saturation will affect the system. I believe that several of these systems will need to be required in small geographical areas in an overall deployment to avoid any scalability problems. Further work will be needed to determine its true scalability and improve on its efficiency with large networks.

The web service could be improved by replacing the Microsoft Access flat file database backend with a SQL based database server which would improve responsiveness under heavy load.

The client currently does not implement cache scavenging, and over time could use quite a lot of disk resources. Further work could be done to prevent this; however this was beyond the original scope of the project.

The UI of the client could be further improved by writing a custom filter for DirectX to allow it to display video in real-time however as this is only a proof of concept it seems outside of the scope.

The networking within the client was designed using a threaded approach, although the implementation could have been carried out without using threads; I have not been exposed to this during my university course.

Acknowledgements

I would like to thank [redacted]

Bibliography

[1] IPTV Global Forecast 2007 to 2011 - Multimedia Research Group Inc. *http://www.mrgco.com/TOC_IPTV_GF1007.html*

[2] Joost – Joost N.V. *http://www.joost.com*

[3] FxCop - Microsoft Corporation *http://msdn2.microsoft.com/en-us/library/bb429476(vs.80).aspx*

[4] Secure Coding Guidelines for the .NET framework - Microsoft Corporation *http://msdn2.microsoft.com/en-us/library/aa302372.aspx*

[5] Subversion - Collabnet *http://subversion.tigris.org/*

[6] StatSVN Repository Statistics - StatSVN Sourceforge Development Group *http://www.statsvn.org/*

[7] Microsoft Virtual Server - Microsoft *http://www.microsoft.com/windowsserversystem/virtualserver/*

All URLs verified as of 19/03/08.